Lecture 8

Brad McNeney

2017-03-07

Grouping

Subgroup summaries

- Data visualization and modelling is often in terms of subgroups.
- Illustrate with some data on enrollments in Stat and Act Sci courses over the 2007/08 to 2015/16 academic years.
  - Data on full-time equivalents (FTEs, equal to 30 credit hours taught) by year and course.
- Recurring theme: Need to split the data into subgroups, transform or summarize, and reassemble, or unsplit.
  - Has come to be known as "split-apply-combine"

Science enrollments database

- Load the `scilong` data frame created by `FTE.Rmd`
  - Look through the `FTE.Rmd` script if you haven't already.

```
library(tidyverse)
load("scilong.RData")
head(scilong)
```

```
##   Subject CrsNum CreditHrs semester enrollment     FTEs year
## 1    ACMA    395         3     1074          1  0.10000 2007
## 2    BISC    100         4     1074         94 12.53333 2007
## 3    BISC    101         4     1074        131 17.46667 2007
## 4    BISC    102         4     1074         99 13.20000 2007
## 5    BISC    202         3     1074        108 10.80000 2007
## 6    BISC    300         3     1074         65  6.50000 2007
```

Stat and Act Sci data

```
stat <- filter(scilong,Subject=="STAT" | Subject=="ACMA")
head(stat)
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year
## 1    ACMA    395          3     1074          1  0.1 2007
## 2    STAT    100          3     1074        123 12.3 2007
## 3    STAT    101          3     1074         48  4.8 2007
## 4    STAT    201          3     1074        134 13.4 2007
## 5    STAT    270          3     1074        114 11.4 2007
## 6    STAT    330          3     1074         29  2.9 2007
```

Split-apply-combine example 1: yearly percent FTEs

- Suppose we want the percent of FTEs in a year that are attributable to each course taught.
- Split the data by year, compute proportion of FTEs for each course in that year, and combine the proportions into a variable that can be included in the `stat` data frame.
- Illustrate base R and `dplyr` approaches.

Example 1: split

- The base R function `split()` splits a data frame on a grouping variable, which is a vector or list of vectors that can be coerced to factor(s), and returns a list.

```
sp.stat <- split(stat,stat$year)
names(sp.stat)
```

```
##  [1] "2007" "2008" "2009" "2010" "2011" "2012" "2013" "2014" "2015" "2016"
## [11] "2017"
```

```
head(sp.stat[["2008"]])
```

```
##    Subject CrsNum CreditHrs semester enrollment FTEs year
## 10    ACMA    210         3     1077         51  5.1 2008
## 11    ACMA    335         3     1077         20  2.0 2008
## 12    ACMA    425         3     1077         22  2.2 2008
## 13    ACMA    465         3     1077         16  1.6 2008
## 14    ACMA    490         3     1077          4  0.4 2008
## 15    STAT    100         3     1077         49  4.9 2008
```

```
str(sp.stat[["2008"]])
```

```
## 'data.frame':    47 obs. of  7 variables:
```

Example 1: Split, cont.

```
sp.stat <- split(stat,list(stat$year,stat$Subject))
names(sp.stat)
```

```
##  [1] "2007.ACMA" "2008.ACMA" "2009.ACMA" "2010.ACMA" "2011.ACMA"
##  [6] "2012.ACMA" "2013.ACMA" "2014.ACMA" "2015.ACMA" "2016.ACMA"
## [11] "2017.ACMA" "2007.STAT" "2008.STAT" "2009.STAT" "2010.STAT"
## [16] "2011.STAT" "2012.STAT" "2013.STAT" "2014.STAT" "2015.STAT"
## [21] "2016.STAT" "2017.STAT"
```

```
head(sp.stat[["2008.STAT"]])
```

```
##    Subject CrsNum CreditHrs semester enrollment FTEs year
## 15    STAT    100         3     1077         49  4.9 2008
## 16    STAT    101         3     1077         59  5.9 2008
## 17    STAT    201         3     1077        284 28.4 2008
## 18    STAT    203         3     1077        164 16.4 2008
## 19    STAT    270         3     1077        185 18.5 2008
## 20    STAT    285         3     1077         47  4.7 2008
```

group_by() from dplyr

- Call is similar to `split`, but we specify multiple variables to group on by comma-separated names.
- Output is a `tbl`. Supposed to be a user-friendly data table. Implementation details not clear.

```
sp.stat.dplyr <- group_by(stat,year,Subject)
sp.stat.dplyr
```

```
## Source: local data frame [524 x 7]
## Groups: year, Subject [22]
##
##    Subject CrsNum CreditHrs semester enrollment  FTEs  year
##      <chr>  <chr>     <int>    <dbl>      <int> <dbl> <dbl>
## 1     ACMA    395         3     1074          1   0.1  2007
## 2     STAT    100         3     1074        123  12.3  2007
## 3     STAT    101         3     1074         48   4.8  2007
## 4     STAT    201         3     1074        134  13.4  2007
## 5     STAT    270         3     1074        114  11.4  2007
## 6     STAT    330         3     1074         29   2.9  2007
## 7     STAT    336         3     1074          7   0.7  2007
## 8     STAT    337         3     1074          7   0.7  2007
## 9     STAT    436         3     1074          4   0.4  2007
## 10    ACMA    210         3     1077         51   5.1  2008
## # ... with 514 more rows
```

Example 1: Apply

- Create a new variable `FTEproportion = FTEs/sum(FTEs)` for each sub-group data frame and save the new variable in the respective data frames.
- Can use the base R function `lappy()`
  - stands for "list apply" – apply a function to each element of a list and return a list as output
- It turns out the following call to `lapply()` does what we want.

```
tem <- lapply(sp.stat,transform,FTEproportion=FTEs/sum(FTEs))
```

- To see why, start with simpler uses of `lapply()`.

Simpler example of `lapply()`

- Define a function to apply to each list element and apply it:

```
fsum <- function(x) { # x is a list element
  sum(x$FTEs) # assumes list elements have an FTEs column
}
tem <- lapply(sp.stat,fsum)
tem[1:2]
```

```
## $`2007.ACMA`
## [1] 0.1
##
## $`2008.ACMA`
## [1] 20.36667
```

Simpler example, cont.

- If our function takes more arguments than just the list element, we add them after the function name.

```
fsum <- function(x,cname) {
  sum(x[,cname])
}
tem <- lapply(sp.stat,fsum,"FTEs")
tem[1:2]
```

```
## $`2007.ACMA`
## [1] 0.1
##
## $`2008.ACMA`
## [1] 20.36667
```

Our use of `lapply()`

- Adding a column to each sub-group data frame requires a function that takes the data frame as an argument and returns the augmented version.
  - This is what `transform()` does

```
head(transform(sp.stat[[1]],FTEproportion = FTEs/sum(FTEs)))
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1    ACMA    395         3     1074          1  0.1 2007             1
```

Putting it all together

```
sp.stat <- lapply(sp.stat,transform,FTEproportion=FTEs/sum(FTEs))
head(sp.stat[[1]])
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1    ACMA    395         3     1074          1  0.1 2007             1
```

Detour: The apply family of functions in R

- The "original" apply is `apply()`, which can be used to apply a function to rows or columns of a matrix.

```
mat <- matrix(1:6,ncol=2,nrow=3)
mat
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
apply(mat,1,sum) # row-wise sums; rowSums() is faster
```

```
## [1] 5 7 9
```

```
apply(mat,2,sum) # column-wise; colSums() is faster
```

```
## [1]  6 15
```

Detour, cont.

- `sapply()` takes the output of `lapply()` and simplifies to a vector or matrix.

```
fsum <- function(x) { sum(x$FTEs) }
sapply(sp.stat,fsum)[1:2]
```

```
## 2007.ACMA 2008.ACMA
##   0.10000  20.36667
```

Detour, cont.

- Other `apply`-like functions `vapply()`, `mapply()`, `tapply()`, ...
- I don't use these.
  - See their respective help pages for information.

The apply step with dplyr

- Actions ("verbs") like `mutate()` are applied to the data within groups when passed a grouped object.
  - That is, the data table is broken into groups and `mutate()` is applied separately to each group.

```
sp.stat.dplyr <- mutate(sp.stat.dplyr,FTEpp = FTEs/sum(FTEs))
select(sp.stat.dplyr,Subject,FTEs,year,FTEpp)
```

```
## Source: local data frame [524 x 4]
## Groups: year, Subject [22]
##
##    Subject  FTEs  year       FTEpp
##      <chr> <dbl> <dbl>       <dbl>
## 1     ACMA   0.1  2007 1.000000000
## 2     STAT  12.3  2007 0.263948498
## 3     STAT   4.8  2007 0.103004292
## 4     STAT  13.4  2007 0.287553648
## 5     STAT  11.4  2007 0.244635193
## 6     STAT   2.9  2007 0.062231760
## 7     STAT   0.7  2007 0.015021459
## 8     STAT   0.7  2007 0.015021459
## 9     STAT   0.4  2007 0.008583691
## 10    ACMA   5.1  2008 0.250409165
## # ... with 514 more rows
```

The combine step

- The base R function `unsplit()` will combine the elements of the list that was generated by `split()`
- Pass `unsplit()` the list of variables used to define the splits.

```
head(unsplit(sp.stat,list(stat$year,stat$Subject)))
```

```
##   Subject CrsNum CreditHrs semester enrollment FTEs year FTEproportion
## 1    ACMA    395         3     1074          1  0.1 2007    1.00000000
## 2    STAT    100         3     1074        123 12.3 2007    0.26394850
## 3    STAT    101         3     1074         48  4.8 2007    0.10300429
## 4    STAT    201         3     1074        134 13.4 2007    0.28755365
## 5    STAT    270         3     1074        114 11.4 2007    0.24463519
## 6    STAT    330         3     1074         29  2.9 2007    0.06223176
```

The combine step with dplyr

- Use ungroup()

```
ungroup(sp.stat.dplyr) %>% select(Subject, FTEs, FTEpp)
```

```
## # A tibble: 524 × 3
##    Subject  FTEs       FTEpp
##      <chr> <dbl>       <dbl>
## 1     ACMA   0.1 1.000000000
## 2     STAT  12.3 0.263948498
## 3     STAT   4.8 0.103004292
## 4     STAT  13.4 0.287553648
## 5     STAT  11.4 0.244635193
## 6     STAT   2.9 0.062231760
## 7     STAT   0.7 0.015021459
## 8     STAT   0.7 0.015021459
## 9     STAT   0.4 0.008583691
## 10    ACMA   5.1 0.250409165
## # ... with 514 more rows
```

Summary of split-apply-combine

- Base R:

```
sp.stat <- split(stat,list(stat$year,stat$Subject))
sp.stat <- lapply(sp.stat,transform,FTEproportion = FTEs/sum(FTEs))
stat <- unsplit(sp.stat,list(stat$year,stat$Subject))
```

- dplyr

```
stat %>% group_by(year,Subject) %>%
  mutate(FTEproportion = FTEs/sum(FTEs)) %>%
  ungroup() -> stat
save(stat,file="statEnrol.RData")
```

Split-apply-combine with `summarise()`

- In the apply step, we may wish to calculate some sort of summary, rather than a transformation of a variable.
- For example, suppose we want to calculate total FTEs by year and subject, and return a data frame

```
stat %>% group_by(year,Subject) %>%
  summarise(totalFTEs = sum(FTEs)) %>%
  ungroup() -> totals
head(totals,n=4)
```

```
## # A tibble: 4 × 3
##    year Subject totalFTEs
##   <dbl>   <chr>     <dbl>
## 1  2007    ACMA   0.10000
## 2  2007    STAT  46.60000
## 3  2008    ACMA  20.36667
## 4  2008    STAT 235.60000
```

Split-apply-combine with `lapply()`

- Compare to base R

```
tem <- split(stat,list(stat$year,stat$Subject))
tem <- lapply(tem,function(x) sum(x$FTEs))
tem[1:4]
```

```
## $`2007.ACMA`
## [1] 0.1
##
## $`2008.ACMA`
## [1] 20.36667
##
## $`2009.ACMA`
## [1] 18.63333
##
## $`2010.ACMA`
## [1] 23.06667
```

- Then would have to write code to coerce output to a data frame.